UNIVERSITY OF CANTERBURY

Scott Ogden 486 end of project report

Creating and evaluating a model for a user in a rehabilitative virtual-reality environment

Scott Ogden

2/7/2014

Scott Ogden[1]
Supervisor: Prof Antonija Mitrovic[2]
Sjo75@uclive.ac.nz[1] | tanja@cosc.canterbury.ac.nz[2]
Computer Science & Software Engineering
University of Canterbury
7 February 2014

Abstract

The ICTG research group is currently working on a Stroke Rehabilitation virtual reality system to improve prospective memory. In this system there is a need to give the user relevant feedback at the appropriate time. To accomplish this, we must model the user's progress and understanding.

Constraint-based modeling was used to create a user model and provide feedback. One of the main design decisions in implementing this was to decide when constraints should be evaluated. Other important design elements included determining what task the user is attempting. As well as writing the constraints, a constraint editor, scripting language and feedback generator were developed. A case study with a stroke survivor was conducted. This case study was used to further develop constraints. Two domain experts evaluated the validity and timeliness of the feedback and the validity of the fidelity of the user model.

Acknowledgements

The Author would like to thank his supervisor Prof. Tanja Mitrovic, as well as Dr. Moffat Mathews for their guidance throughout the project, as well as their remarkably thorough feedback and advice. To Jay Holland and Anthony Bracegirdle, thank you for your help with Unity and the VR environment. This report would not have been possible without the help of the CSSE Department's Summer COSC486 Scholarship.

Contents

1. Introduction

1.1. An overview of the Research

In this project the author modelled a user's progress through a virtual environment in such a way so as to determine whether the user was successful and to give them an appropriate level of feedback. It is part of a larger project that attempts to improve a user's prospective memory by presenting scenarios of various types to users. They will be taught memorisation techniques and then taken through the scenarios in a virtual reality environment. The scenarios vary in complexity due to the subtlety of cues and the concurrency of tasks.

While a number of approaches to user modelling were considered, constraint-based modelling was the method perused. Constraint-based modelling involves sets of constraints, which should always hold true while the participant is acting correctly. The virtual reality environment (described in the next section) transmits the user's actions to the server which evaluates the actions against the constraints. The system can then create the model to determine the success of the participant or what they are failing to do. In this way we can convert the user's actions into real-time feedback. The hypothesis for this project is: *it is possible to model a user's progress through everyday tasks and evaluate the progress in real time.* To test whether the hypothesis holds and if the user can be modelled in this way, two domain experts evaluated the system and compared their expected feedback with the feedback generated.

1.2.Stroke rehabilitation project

This project is part of a larger research effort. The larger project is based around improving the perspective memory of a stroke survivor through a number of memorisation techniques. It has been well established in the past that sufferers of brain injury often suffer from decreased prospective memory function [1] [2] [3]. A 3D virtual reality environment has been created in Unity3D to give the users a safe environment to practice these techniques. The user will be given a number of tasks to complete (more information about tasks are in section 3.2), some of which will be concurrent, and cues can become active at any time. Having more than one task at once will make the activities considerably harder for the user as they will have more information to remember and process [4].

The user is able to carry out a number of actions on a variety of objects such as clothes-lines and radios. Some of these actions require various other inventory items to be collected before-hand; the user is able to view their inventory at any time. Some tasks are time dependent and the user is able to view a clock whenever they choose. Before the user enters the virtual environment, they are prompted for their name and then given the list of tasks to memorise. After they choose to exit from the environment, statistics about their actions are displayed.

Virtual environments have been used extensively in a variety of Neuroscience contexts [5] [6] [7]. Many of these virtual environments for Neuroscience have been specifically for stroke rehabilitation [8] [9]. While projects in the past have used a virtual environment to assess a user's memory [10] [11], this project is unique in how it attempts to restore the patient's prospective memory abilities.

Figure 1: The Virtual reality environment, displaying some feedback.

1.3.Motivation for the Research

It was suggested in Olsson's seminal paper on Learning from Performance Errors [12] that constraint-based feedback would be beneficial for education. By being able to provide accurate and timely feedback, we hope to quickly change the user's behaviours, as well as to help them to complete the simulation. It is also important that we have an accurate User Model. This will be beneficial, as it will help inform future experimenters as to what users do or do not understand, and will help with task selection.

1.4.The Structure of the Report

Section 2 of this report contains background information directly pertinent to the system. It also presents relevant background in sufficient detail to support the comprehension of this report. Section 3 first explains the goals and hypothesis of the experiment, followed by a comprehensive description of the key design and implementation details for the system developed. Section 3 also contains details relating to the pilot study conducted. Section 4 explains how the hypothesis was evaluated, and the results of that process. Section 5 is the conclusion, yet also encompasses details such as future direction. This is followed by a bibliography and appendices.

2. Background research and context

2.1.Intelligent Tutoring Systems

Intelligent Tutoring systems are an effort to use computers to teach humans. The end goal of such an endeavour would be for the computers to be as effective as a human tutor [13]. Intelligent tutors cover a wide range of topics from LISP and programming skills [14] through to chemistry [15].

There are a number of reasons as to why this would be useful, and indeed how an ITS could be more effective than a human tutor. "For learning to be effective, the student must be active and have opportunities to practice important skills. Intelligent Tutoring Systems (ITS) provide tailored support to each individual student, a unique strength that comes from their ability to model various types of knowledge required for instruction: domain knowledge, knowledge about their students (i.e. student models), models of pedagogy and communication knowledge" [16].

2.2.User modelling and constraints

In the past Student (User) modelling has been defined as: “the process of gathering relevant information in order to identify the knowledge state of a student. In an ideal case, the model of a student should illustrate his/her knowledge, preferred learning strategies, areas of interest besides that of instruction, preferred presentation style, and so on” [13]. The ability to model the user’s performance and adapt is pivotal to an Intelligent tutoring system.

User models can be executable or may be rules or even schemas. However they are prepared, they are inherently complicated, and must be precise enough to be evaluated. A massive amount of knowledge elements are required to create a remotely accurate model. It is impossible to fully model the user, that level of detail is simply unavailable. We must instead make approximations about the user and make decisions based on that. This lack of detail has been referred to “as the oversimplicity problem”. It should be noted that the usefulness of a fully detailed user model has also been debated, as Sandberg noted: "detailed user models do not necessarily enhance the capability of an intelligent tutoring system ... good teaching can do without a detailed user model, because in good teaching serious misconceptions are avoided, and errors will be repaired on the spot ... it is debatable whether the cost of constructing very detailed, complex user models that are runnable and have to be maintained all the time is worthwhile in terms of the gain in teaching efficiency” [17].

Constraint-based modelling is a student modelling technique which represents the domain knowledge a set of constraints – basic rules required for a solution to be correct. It has been found to be very effective for creating tutors and learning tools in the past [13]. Constraints were conceived by Olsson as a knowledge-representation formalism resulting from his theory of learning from performance errors [15] [12]. Constraints have two components: a relevance condition and a satisfaction condition. The relevance condition details when the constraint is relevant, while the satisfaction condition details what must be true for the constraint to be satisfied [12] [16] [18]. It has been frequently described as: “If <relevance condition> is true, then <satisfaction condition> had better also be true, otherwise something has gone wrong” [18] [16] [19].

Unlike one of its main competitors, model tracing, constraint-based modelling focuses on solution states rather than transitions between states. If the user reaches a state that violates a constraint, it becomes impossible for the user to reach the desired end state. “Constraints support evaluation and judgment, not inference, and are used to represent both domain and student knowledge” [16]. Because constraints are more abstract and only compare the states, rather than any possible action, it is normally the case that they are easier to create and maintain than the alternatives [18].

In the past, constraints have been written in a language which could easily evaluate textural constraints, such as LISP. An example of this design choice would be SQL-Tutor [20].

“constraint-based modelling uses abstraction to avoid the need to model students’ misconceptions. Constraints represent only correct knowledge in terms of pedagogically significant states; each constraint maps to a set of solution states that share the same domain principle. A constraint-based ITS can therefore react in the same way (e.g. by displaying the same feedback message) for any solution that violates a given constraint” [16].

After a constraint has been evaluated, it is added to a list of relevant, satisfied or violated constraints as appropriate. These lists of constraints comprise the short term model [18] [16] [19]. These short term models are combined into the long term model. Many opposing methods have been suggested to represent the knowledge of the student or user. Some examples are: an overlay on top of the domain model, as a set of performance histories for all constraints used by the student, or even a Bayesian student model [16].

2.3.Feedback

If a constraint is violated, the user needs some means of knowing that he/she has made a mistake, and they need to know what needs to be done differently next time. This is the role of feedback.

The operation of feedback was neatly summarised in [16]: “Pedagogically CBM determines the content of instruction provided. If there are errors in a student’s action, the ITS will present feedback provided by the violated constraints. The form of this feedback is shaped by the underlying learning theory: it should tell the student what domain principle he/she has violated, how it was violated by the student’s solution, and reiterate the correct domain principle. However, the style and delivery of feedback is independent of CBM; they can be adapted to a particular student. For example, feedback can be given in textual and/or pictorial form, depending on the student’s learning style. Further, both the timing (immediate or delayed) and amount of feedback (i.e. the number of feedback messages and the level of detail) can vary and can be adaptive.”

In a Model Tracing ITS, feedback is normally given immediately, after every action the student performs. This is not as common in Constraint Based ITSs. Constraint-based ITSs compare states, and so feedback is typically given after each submission [19]. In this way the user is given more freedom to experiment before being instructed on what they should be doing. When comparing constraints, a situation may arise where multiple constraints have been violated. SQL-Tutor uses the student model to determine which violation should be targeted for teaching through feedback. The user will be informed of the feedback and also of the number of constraints violated. The rationale for this is that “it is easier for the student to work on one error at a time and that multiple feedback messages related to multiple errors or misconceptions might be confusing or overwhelming” [19].

There are however other approaches to feedback. In [20], to avoid the “, 'tutor knows best' style of ITSs which alienates classroom teachers”, the ITS constructs a solution from the user’s answer and the violated constraints. The resulting solution is correct while still being similar to the answer they had attempted.

2.4.Adaptivity

In a simple learning model it could be that: on the first submission, the user is given only basic information which increases for every breach of the constraint. For example, in [16] : “For the first submission, the student is only told whether or not the solution is correct. If they continue to submit incorrect solutions, they will progressively get more help until they are able to complete the problem or ask for the solution. The student model is used to select the next problem to be posed to the student, and we have experimented with a number of different problem-selection strategies.”

Adaptability is important as it allows a more fine grained understanding of how much help the user needs and so supports a more precise student model.

3. The work

3.1.Goals and Hypothesis

There are several goals for this research relating to user modelling in the stroke system:

1. Establish the feasibility of creating a user model in the Stroke Rehabilitation Virtual Environment
2. Evaluate the usability of constraint-based modelling in this context
3. Develop software to process user actions into feedback
4. Develop the constraint set
5. Record the users breaches or otherwise of constraints in the user model.

The Hypothesis to be tested by this research is: *It possible to model a user's progress through everyday tasks and evaluate the progress in real time.*

3.2.Tasks

The users are given a number of tasks to memorise and then conduct in the virtual environment. A task can comprise multiple actions, on multiple objects in multiple rooms. To give one example: *When the washing machine stops: take the dress from the washing machine, and hang it on the clothes line*. All tasks have a cue and a number of steps to complete. Tasks either have time triggered cues, or event triggered cues. These two types of cues reflect the cues identified in [21], although additional cues have been proposed such as activity based cues [22]. All tasks contain inventory steps and/or normal steps. Tasks should only be attempted from after a point known as 'cue discovery'.

Tasks with time based cues are 'discovered' several minutes before the stated time. For example, if the task is *turn on the radio at 6.00pm*, the user can start to move towards the radio and interact with it a few minutes earlier in preparation. Event-based cues only begin when the stated event occurs. Consider the task: *bring in the washing when it starts raining.* For this task, the user has no way of knowing when it is going to rain, and so they are not eligible to begin the task before the cue is discovered.

Inventory steps involve the collection of various collectable inventory items. Normal steps are steps involving interacting with a specified object in the game world. Consider the task *take the white dress and iron it*. The first step involves collecting the inventory item *white dress*, this is an inventory step. The second step involves operating the ironing board, and so it's a normal step. Each normal step has exactly one gameObject (that is the object to be interacted with) as well as a specified interacted. Tasks and their composition are important to consider when interpreting the constraints.

For every task, in real life as in the virtual reality simulator, there is a finite amount of time for which the task can be completed before it becomes obsolete/impossible. However, this alone is not the only factor in determining which tasks are more important. Some tasks, such as turning off the stove, have worse outcomes for failing to complete than other tasks do, such as turning on the Television.

Initially to simulate time constraints and priorities each task was intended to have a 'priority curve'. As time progressed, the level of importance of the task would change along the curve. It was later found that this could be greatly simplified, with only minimal concessions. Now tasks are created

with a priority level, from which a valid completion window is generated. Priorities range from 0 to 5, with 5 being the most important. Tasks with a level 5 priority are tasks with a very real chance of injury of household damage if they are not completed on time. Tasks with a priority of 0 are unimportant. It was decided to give priority quite a coarse scale, as it is a subjective topic, and we did not what users unduly punished for having differing yet reasonable judgements. A typical priority 5 task would be: *when the timer beeps, turn off the stove top*. By contrast, a priority 0 task may be: *When you are finished all other tasks, watch television.*

From 'Cue discovery', the user has (6-priority)x4 minutes to complete the task before it becomes obsolete. The four minute band size was chosen arbitrarily and can be increased or decreased to change the difficulty of the simulation. For time-based cues, discovery occurs a few minutes before the prescribed time. This is so that the user can begin moving towards the tasks before it is relevant without being reprimanded. For event driven cues, such as 'when it begins to rain' there is no lead in time, as the user should not be able to predict when this is about to happen.

There are three constraints that involve certain amounts of time. Tasks with only one minute left should be done before tasks with more than one minute left, even if that task with more than one minute left is of higher priority. In this way the user can still complete all the tasks. It is also important to bear in mind that higher priority tasks will reach the point of only having one minute left a lot sooner than a lower priority task. If there are multiple tasks with less than one minute left, they should choose the highest priority one. The user also breaches a constraint if they let any constraints come so close to invalidation. When that constraint is breached they are also given feedback on what task they should attempt urgently.

The next threshold is at five minutes. Users must do tasks with less than five minutes left before they attempt tasks with more than five minutes left. To summarise, tasks are first stratified according to time left into less than one minute, less than five minutes, more than five minutes. From there they are ranked according to priority. If any tasks have equal time strata and priority than they can be done in any order, otherwise they must pick the top one.

3.3.Logical Conditions

In interpreted programming languages, such as LISP, it would be sufficient to have constraints typed in the language of the system, and then evaluated in run time. However, Java is a compiled language, and does not support evaluating strings as commands. As such it was necessary to make some form or structure for inputting and comparing these logical conditions.

Each logical condition has on comparator, for example 'AllEqualTo', and then two arguments. It can also be set to be inverse. *Table 1* discusses the various comparators and their uses, while *Table 2* discusses the values and their interpretation.

Comparator	**Use**
On Route to	Uses Dijkstra's algorithm to calculate if argument one is on any path to argument two
On All Routes To	Uses Dijkstra's algorithm to work out if it is possible to get to argument two without going through argument one
In List	Does argument two contain argument one
One Equal to (Pairwise)	Pairs off the values in argument one and

	argument two. Returns true if there are any matching pairs.
One Lesser than (Pairwise)	If any-one value from argument one is lesser than the value in the same position in argument two returns true
One Greater than (Pairwise)	If any-one value from argument one is greater than the value in the same position in argument two returns true
One Equal to	If any element from argument one is equivalent to a value in argument two then this is true.
One Lesser than	If any element from argument one is lesser than a value in argument two then this is true.
One Greater than	If any element from argument one greater than a value in argument two then this is true.
All Equal to (Pairwise)	Compares two lists and checks that they are the same, in order and in values
All Lesser Than (Pairwise)	Pairs off values in order from the two arguments, requires that the values from the argument one side of the pair are lower
All Greater Than (Pairwise)	Pairs off values in order from the two arguments, requires that the values from the argument one side of the pair are greater

Table 1: Comparators present in the system and their applications

Value	**Choices**	**Explanation**
List	Any of the lists exposed by the system	Returns an ArrayList with the appropriate values
Game Object	Previous / Current	Returns the name of the object that was/is being interacted with
Room	Previous / Current	Returns the name of the room that was/is being interacted with
Value	User input	Returns the value entered. Can be cast with \d for Double, \b for Boolean, \l for Long, or \i for integer. By default it is cast to String.
Task detail	User input	Not implemented
Variable	Any of the variables exposed by the system	Returns an object of the appropriate value
Distance To	User input	Returns the number of nodes from the user's current room to the user provided room
Special list	User input	Returns a value using the scripting system detailed in the next section

Table 2: Types of values

Most variables can be defined in advance. Unfortunately some operations, particularly those using comparisons are too complex to easily define options for in advance. For that reason a lightweight

scripting language is provided. This has two key uses. Firstly it is used in the feedback messages to make them more specific to the current task. Secondly, it is used in the 'special list' argument in logical statements to provide a more powerful set of evaluation options.

There are six main symbols used in this scripting language: '.', '|', '<', '#', '$' and '()'. Almost all statements begin with a '|'. This symbol instructs the system to lookup the variable or list between the '|' and the next symbol. If it is a variable it is then converted into a list with only one entry.

To perform an operation on these values, there are two options. The '.' uses reflection to use the next segment on all the previous values. For example, consider the statement '|goalObjects.getRoom'. This will iteratively execute getRoom on all of the goal objects, and return a list of those rooms. In a case where a list is returned by a method, such as .getInventory, it will by default create a list filled with those lists. If however a '<' is used, it adds all the values to one general list, with no sub lists. Some methods require arguments, in these situations, a '#' followed by a number can be used for numerical arguments, or a '$' followed by a String for String arguments.

Lastly are the '()' commands. These do not act like normal parenthesis. The condition inside of the bracket is evaluated for every value in the list, and it returns a list where only the values from the original list for which the statement is true are present. Perhaps this is best explained with an example: '|goalObjects(.requiresCrouch is oneEqualTo value true\b)'. First the goal objects is evaluated and returns a list of all the goal Objects. Then for each of those objects, the result of the requiresCrouch method is compared to the Boolean value true. If it is true, then the original goal object is added to the list to be returned. The logical statement is written in the form: '([arg1] is[n't] [comparator] [value] [arg2])'. Argument one is always a special list, but argument two can be of any defined value.

3.4.Feedback

Let us discuss further the feedback and learning strategy. Every time the user violates a constraint, this is recorded in the student model and feedback is generated. Every violation leads to an escalation in the level of feedback – originally this was not the case, it could be that each feedback level lasted an arbitrary number of breaches, but this was determined to be not as effective. For more information on strategy selection please see the background information section. The levels of feedback always go from the most general, vague feedback, through to the most specific. To give an example, for the constraint that ensures that the user is going to the right location, the first feedback message is: "You're going the wrong way!" If they continue down the wrong path, they will get the more specific "perhaps you should be going to the [|goalRooms]". This cumulates on their third breach with "perhaps you should be going to the [|goalRooms] and use the [|goalObjects]". This is the most specific and reminds them of the task at hand. If the user continues to violate this constraint, they will continue to get the most specific level of feedback, and the breaches of the constraint will continue to be recorded.

Each constraint can be configured to ‘reset’ back to the most general level either at the start of a new task, the start of a new session, or never. Feedback similar to the example above would be more appropriate for the start of a new task, as the old feedback information they were given may be out of date. Session specific constraints are generally more related to game skills. For example, a constraint which reminds them how to interact with objects if they have been staring at objects for too long will only play the first few times this happens in a session, to avoid annoying the user or overwhelming the user.

Figure 2: The virtual reality environment displaying feedback.

In addition to the feedback being displayed during the session, the user can also push the ‘H’ key for more help. If the user has had a message displayed in the last 30 seconds, this message is displayed again to remind them of what they were doing wrong. Otherwise the default message is displayed. If there are no tasks left to do, the default feedback informs them of this. Otherwise it gives them increasingly specific hints as to what they should be doing.

3.5.Editor and Constraints

As the Author will not be involved in the maintenance of neither the program nor the constraints, it was important that they both be easy to edit. Many other constraint based tutors, such as SQL Tutor have the constraints written as executable code. This is extremely flexible and extensible, but can lead to more time spent debugging, and a much steeper learning curve. This project uses a point and click constraint management system. A novice can create basic constraints without any knowledge of the underlying system, but it is extensible as a more advanced user is able to use the implemented scripting language to reflect variables used by the system. In this segment we will discuss the various parts of the editor. In doing so, we will also describe and discuss how constraints are implemented in the system.

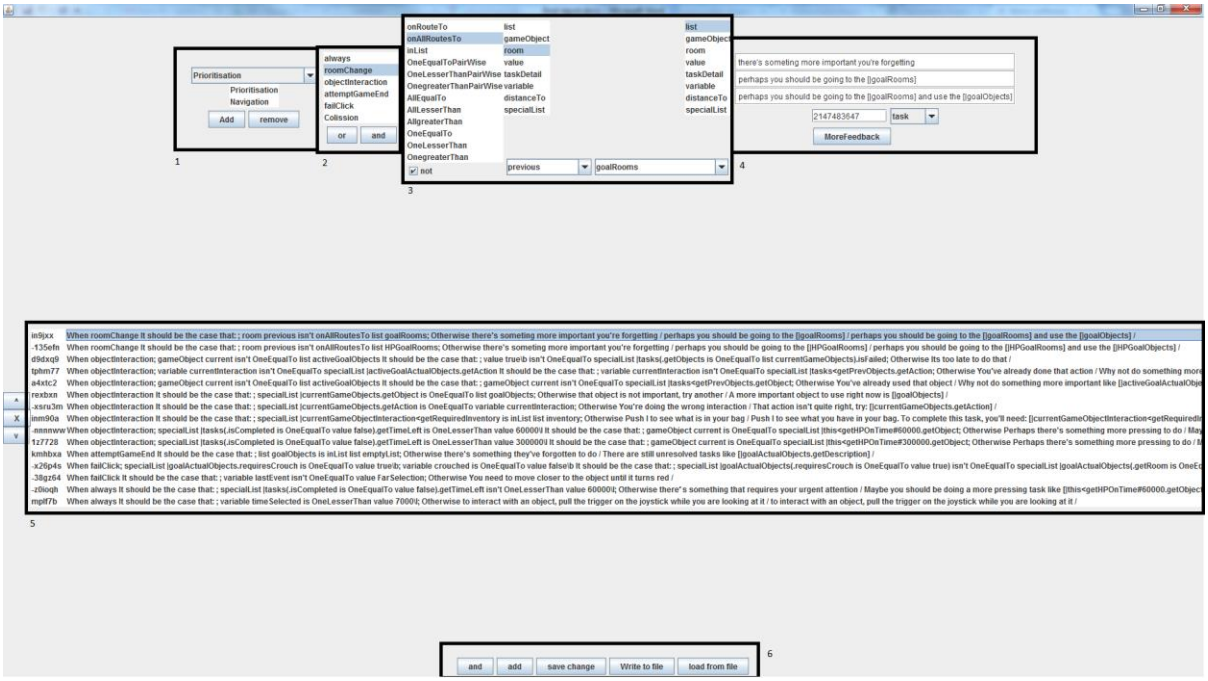

Figure 3: The Editor (numbered Boxes added for explanation)

Box one shows the ontology of the constraint. Every constraint can have an arbitrarily large number of areas in the ontology which it relates to. In the example above, it is showing that this constraint (which is about going to the correct room) is relevant to prioritisation and navigation. Constraint Ontology is covered more in depth in a previous section.

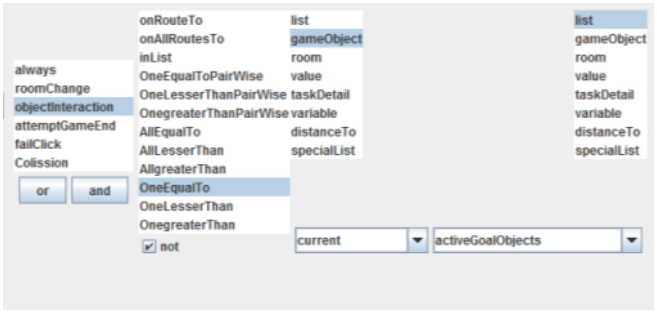

Figure 4: The relevance condition for the constraint.

Box two shows the relevance condition for the constraint. In *Figure 3* only the most basic type of relevance condition is shown. That is to say it has one of the relevance triggers discussed above only. *Figure 4*, a relevance condition with a logical condition is shown. Each logical condition is either a logical and, or logical or.

Figure 3, box three shows the satisfaction condition for that constraint. As with relevance conditions, further 'logical conditions' can be added. We will discuss the nature and compositions of a logical condition in a later section. The main benefit to having this point and click selection rather than just using the scripting language is that since the variable are automatically suggested and linked, most of the time reflection or text comparisons can be avoided, thus greatly reducing the development and testing time of new constraints.

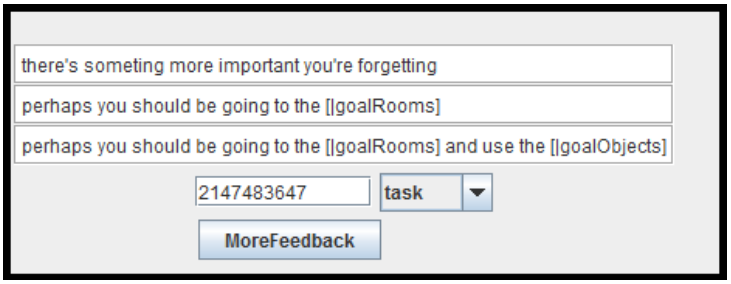

Figure 5: The feedback for the constraint.

Figure 5 shows an enlargement of the feedback segment of the editor. Each line of feedback is one step, starting with the top one and working through

to the bottom. As seen in *Figure 5*, feedback messages can be dynamic; any information located inside of [] will be rendered in the scripting language. There are variants on the normal scripting language to support a more natural English response, such as [+] which switches list separators from 'or to 'and'. We will discuss the scripting language in a later segment. The step which the constraints feedback is on can be reset on a 'submission', the end of a session, or never. This is controlled by setting the feedback as either 'task' 'session' or 'allTime' respectively. The number represents the maximum number of times to display the feedback before a reset (or enter a -1 for the maximum).

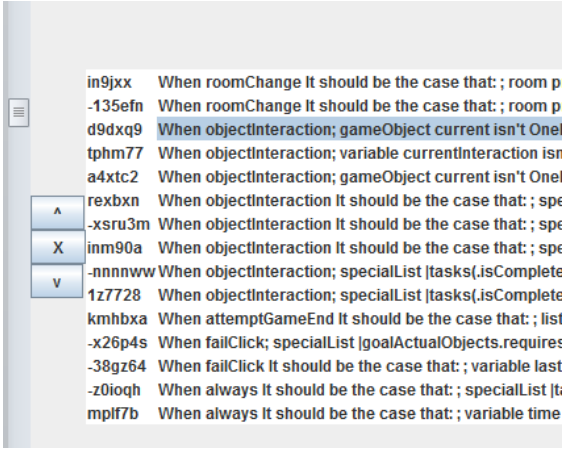

Figure 6: Existing constraints

Box five of *Figure 3*(and its enlargement (adjacent)) shows the existing constraints in vaguely natural language representation: 'When *relevance condition* it should be the case that: *satisfaction condition*; Otherwise *Feedback"*. Importantly, next to the constraint is its randomly assigned hashing code. This is the code that the constraint is identified by in the user model. As a constraints natural language representation is extremely verbose, it is more useful to refer to them in this short form. Other similar tutors use an ID number, but a hash has been used here to make the codes easier to distinguish, and so that if a constraint is removed and later added back, the user model will have the same hash, and the record can continue.

Lastly, box six of *Figure 3* shows some of the typical editor functions, such as adding more conditions, adding the constraints, saving loading etc. Rather than saving the constraints in plain text, they are stored in a serialised format for integrity.

3.6.Developing constraints

The generation of constraints has several stages and sub stages. The first of these stages is to generate a list of the types of tasks that would be completed by the user. We needed an idea of the variety of tasks for which there will need to be constraints. This step was aided by the author's previous unpublished work [23], it was also aided by the list of tasks already programmed into the virtual reality environment.

The next step was to decide what could be done in the course of completing the task that would be incorrect: for example, going the wrong way, or choosing the wrong task. Initially these decisions were made on the basis that only one task would be active at a time, but later iterations added support for multiple simultaneous tasks. From the basic rules, the log files and the source code of the virtual reality environment were evaluated to determine which indicators would be available to determine if these rules were broken. From these indicators constraints were derived.

These initial constraints underwent three rounds of expert evaluation. A number of other constraints were added at the recommendation of the experts. Some of these were to help with game skills, requiring that the user is crouched, if they need to be crouched to complete the tasks in the room.

Others were regarding timing/prioritisation or to do with selecting correct actions. Other constraints were refined and one, which was a 'negative constraint', was removed.

Later constraints were also informed by the pilot study (see section 3.11). This pilot study demonstrated what issues could be arrived at in a real world scenario and provided the opportunity to create constraints to address these issues ahead of time.

Fifteen constraints were developed for use in this system. Before we begin to list and explain the constraints, we must first discuss some of the vernacular used in the listing. While terms used in relation to constraints in general are discussed in background section of the report (section 2), terms unique to the constraints for this environment are as yet unexplained.

Goal object refers to any object which is part of at least one currently active task. It does not include objects which have already been used, unless they are as yet unused for some other task. Every goal object has a room it is located in (goal room) and exactly one interaction required to complete it(foal interaction). If multiple interactions are required, these are stored as separate goal objects. Collectively, the goal objects which relate to tasks of the highest priority rating currently active are the High priority goal objects. The most important goal object is a single object. It is selected using the criteria described in the section earlier in this report relating to timing and priority. Only a single object is selected, even if it is a tie for highest rank. This is to make matters easier for the users, rather than make them choose between multiple objects.

Other than those terms, *Table 3* also makes use of the scripting language which was explained in an earlier section, and the feedback step up/reset concepts explained in the feedback section.

Skill Areas	Relevance Condition	Satisfaction Condition	Feedback (levels separated by a '/')	Feedback resets on	Purpose of constraint
Navigation	When the player moves room	The room they are coming from should not be on all routes to every goal room	You're going the wrong way!/ perhaps you should be going to the [|mostImportant. goalRoom]/ perhaps you should be going to the [|mostImportant. goalRoom] and use the [|mostImportant. goalObject]	Task	To ensure that the user is not completely lost
Prioritisation, navigation	When the player moves room	The room they are coming from should not be on all routes to every High priority goal room	there's something more important you're forgetting/ perhaps you should be going to the [|mostImportant. goalRoom]/ perhaps you should be going to the [|mostImportant. goalRoom] and use the [|mostImportant. goalObject]	Task	To ensure that the user is going in the most appropriate direction
Miscellaneous	When the player	The object they are	It's too late for that	Task	To inform the user that it is

	interacts with an object that is not a current goal object	interacting with should not be expired			too late to attempt that task.
Remembering what has been done	When the player interacts with an object and the action is not a goal interaction	The interaction should not be a previous goal interaction of the object being interacted with	You've already done that action / Why not do something more important like [|mostImportant. getDescription]	Task	So that the user does not get confused between the different interactions when more than one task involves that same object
Remembering what has been done	When the player interacts with an object that is not a current goal object	The object being interacted with should not be a previous goal object	You've already used that object/ Why not do something more important like [|mostImportant. getObject]	Task	So that the user realises that they have already done that task/sub tasks, and does not get stuck in a loop of using the same object
Remembering Tasks Correctly	When the player interacts with an object	The object being interacted with should be a goal object	that object is not important, try another / A more important object to use right now is [|mostImportant .goalObject]	Task	This constraint is important so that the user remains on task
Remembering Tasks Correctly	When the player interacts with an object	The interaction the user is doing with the object should be a goal interaction	You're doing the wrong interaction / That action isn't quite right, try: [|currentGameObjects. getAction]	Task	Helping the user complete a task, if they have forgotten what they are meant to do with an object
Remembering Inventory	When the player interacts with an object	The player should have the required inventory for that interaction with that object	Push I to see what is in your bag / Push I to see what you have in your bag. To complete this task, you'll need: [|currentGameObject Interaction <getRequired Inventory]	Task	To help the user work out what inventory they are lacking to complete a task

Prioritisation	When the player interacts with an object and there is a task with less than 1 minute left	This object should be the highest priority object with less than a minute left.	Perhaps there's something more pressing to do / Maybe you should be doing a more pressing task like [|mostImportant. getObject]	Task	See section 3.4
Prioritisation	When the player interacts with an object and there is a task with less than 5 minutes left	This object should be the highest priority object with less than five minutes left.	Perhaps there's something more pressing to do / Maybe you should be doing a more pressing task like [|mostImportant. getObject]	Task	See section 3.4
Remembering All Tasks	When the player attempts to end the simulation	There should be no remaining goal objects	there's something you've forgotten to do / There are still unresolved tasks like [|mostImportant. getDescription]	Session	Gives them the chance to complete all tasks, if they have forgotten about some at the end of the simulation
Game Skills	When the player clicks and there is no object selected, and there is at least one goal object which requires crouching and they are not crouched.	None of the goal objects in the room should require crouching	you may need to crouch. push C to crouch / if you want to [|goalActualObjects (.requiresCrouch is OneEqualTo value true) (.getRoom is OneEqualTo variable currentRoom). getDescription] you will need to crouch	Task	Reminds them to crouch if they forgot that some actions will require crouching
Game Skills	When the player clicks and there is no object selected	You should not be looking at something but a little too far away for it to be selected	You need to move closer to the object until it turns red	Session (maximum 3 displays)	Remind users to get close to objects to select them

Prioritisation	Always relevant (checked every 500ms)	There should be no tasks with less than one minute left	there's something that requires your urgent attention / Maybe you should be doing a more pressing task like [|this<getHPOnTime #60000.getObject]	Task	Helps the user avoid task expiry
Game Skills	Always relevant (checked every 500ms)	They should not have selected an object for more than 7 seconds	to interact with an object, pull the trigger on the joystick while you are looking at it	Session (maximum of 3 displays)	To remind users of how to interact with objects

Table 3: The constraints present in the system

3.7.User model/Submissions

There are several differences in this system compared to other constraint based tutors. Whereas others such as NORMIT, SQL-Tutor or EER-Tutor tend to evaluate the constraints at the same time as they are recorded in the user model, this system has to separate the two roles. An Intelligent tutor such as SQL-Tutor has a very clear submission point where the user makes it clear that they believe they have solved the problem or when they would like to get feedback. At this point the constraints can be evaluated, and any constraint that has not been violated can be marked as such. In this tutor, we have no way of knowing which task they are working on, and there is also the possibility that they are switching between problems. When they go into the wrong room, it does not necessarily mean that they know which task to prioritise, but it does definitely mean they're going the wrong way. Constraints are evaluated when there is potential for their relevance conditions to be true. In *Table 4* you can find the relevance conditions and examples of constraints which would have these relevance conditions. In addition to the relevance conditions in the table, relevance conditions can also have logical statements, such as: 'crouched is not equal to true.'

Relevance Condition	Example constraint
Every 0.5 seconds	It should be that no tasks about to expire, otherwise you should be working on those tasks
Changing room	If they are changing room, it should be that they should be moving towards a task
Interacting with an object	If they interact with an object, it should be that the object is related to one of their tasks.
Attempting to end the simulation	When they attempt to end the simulation, it should be that they have finished all their tasks
Clicking when no object is selected	When they fail to interact with an object and they are not crouched, it should be that none of the important objects in the room require crouching.

Table 4: The relevance conditions

As mentioned above, when one of these relevance conditions are met, and the satisfaction condition is not met we know they have done something wrong, but have no way of knowing which task it relates to. We know when to record a failure, but not when to record a success. Thus there

situations were arrived at where we record the success, i.e. not breaching a constraint. These three situations are: completing a task, finishing the simulation and the most important task changing.

When a task is completed, we know that every breach between this and the last 'submission' is likely that has related to completing this task. This is not necessarily the case, but it is a good approximation. We can thus say what every constraint not breached since the last submission is a 0 in the user model, and every other constraint is a series of 1s depending on how many breaches were made. We know that the user may switch between tasks, but it is important that they work on the most important one at any time. To that end, there are a number of constraints based around prioritisation of tasks. When the most important task changes (i.e. a task is about to expire, or the cue for a more important task has been discovered) the user should switch tasks. We do not know what task they were working on, but it is likely to be different to the task they are now working on. For this reason, we now count this as a 'submission'. It is recorded which constraints were or were not breached in the process of working on the previous task. When the user ends the simulation, this is also considered to be a 'submission' as we know from this point on they are unable to complete any more tasks, or breach any more constraints.

Each user has their own user model. When a new session in the virtual reality environment is created, the user model is fetched or created as appropriate on the server side. Whenever the user attempts to end their session, the long term user model is exported as comma separated values into an easy to read file. Each constraint is represented as one line. The line contains first the ontology information, i.e. what skills does that constraint relate to, followed by a hash of the constraint then the string of 1s and 0s representing the users past performance at that task. Constraints in their written form are long and unwieldy, and so hashes are used instead to quickly relate the constraint to the user model. The other benefit of using hashes instead of the actual constraint object is so that if constraints are added or removed, or even removed then returned, the user model can still continue intact.

It is very important that we keep track of the constraint violations relating to the current task, as we rely on this knowledge for deciding what feedback to give, however this was be discussed on the section relating to stepped feedback and adaptively.

3.8.Network/Deployment

Initially it was intended that the feedback generator would read the log files of the virtual environment, process the information then write the feedback into another log file which would be read by the virtual environment. However, this solution would not scale well when eventually it is deployed to a server and connected to many virtual environments on other computers. Additionally, there was the possibility of read/write conflicts.

It was quickly decided to migrate to a network solution with the two programs communicating with network packets. This was made difficult by the differing programming environments. The virtual environment is programmed in Unity, which is able to implement some C# code, but does not use the ordinary Microsoft .Net framework; instead it uses a partial implementation of Mono. Unfortunately there is no support for threads. The other issue that arose from this was involved NAT push-through. When the feedback generator and the virtual environment operate on the same computer they can communicate without a problem. However when run on two separate computers in such a complicated network as the University of Canterbury, it is impossible for Unity to receive

the responses from Java. To combat this, the virtual environment must be deployed with a custom built 'network helper' which communicates locally to the virtual environment and over the network to the central server. See *Figure 7:* for the deployment diagram. It should be noted that the user model, constraints, and so on are all stored on the central server, and are part of the feedback generator program.

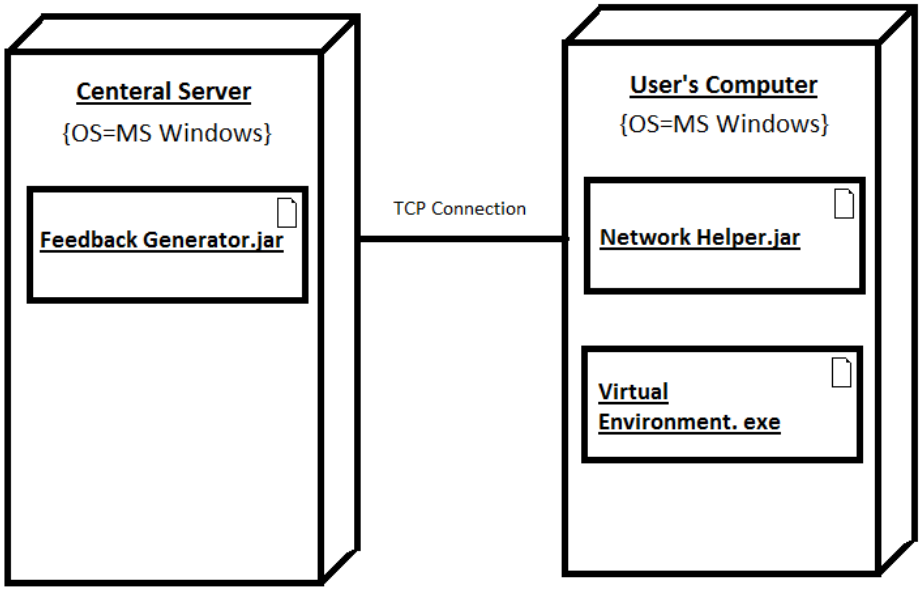

Figure 7: Diagram depicting the deployment of the system across different computers

3.9.Case study

Stroke patients can have a number of impairments which make operating a system such as this considerably more difficult. Before the commencement of the research, the virtual environment had never been tested with a stroke survivor before. On Wednesday the 8th of January we tested the system on a stroke survivor. He was given 14 practice tasks, which he was helped with, to familiarise himself with the virtual environment and the controls. This was followed by 4 more complex tasks. In addition he was given a map and basic instruction on how to control the environment. See *Table 5* for the list of practise and actual tasks:

Practice Tasks	**Tasks**
Walk over to the fish	At 5:55pm turn on radio so that you can listen to the racing results.
Feed the fish	Once you've heard all the racing results, feed the fish
Open the front door.	When the temperature drops below 18 degrees, turn on the woodburner.
Go to the lounge	At 6pm, put on the kettle.
Turn on TV	
Check the temperature	
Go to Kitchen	

Check time	
Go to Laundry	
Go to clothes line	
Take clothes	
Go back to Laundry	
Lock door	
Go to living room	

Table 5: Tasks from the case study

The user has severe Dyspraxia and Aphasia, so was not able to give us clear feedback directly on what he found good or bad about the system. The various programs recorded all of his actions in their log files, and we were able to observe what tasks he spent longer on, or found more difficult.

Unfortunately he did not even notice the feedback box (see *Figure 2*), but would have been unable to read it anyway. The target user for the VR environment would be more literate. This however led to an idea for future development of the system, in that it could speak the feedback out loud, to make it easier for stroke survivors who have difficulty reading. Several other ideas were generated and implemented: several new constrains were added to help users struggling with basic game skills, such as not properly selecting objects; some minor bug fixing was implemented; and more constrains to help the user understand what they should be doing was added.

The User also had a number of other difficulties. Much of the information was presented in a difficult to read fashion, due to font size, colour, display time etc. Also, the constraints where the cues do not link specifically to the goal object were very difficult. To give an example: 'When the temperature drops below 18 degrees, turn on the woodburner.' This is difficult as when looking at the thermometer to determine if it is below 18 degrees, the wood burner is out of sight. It was difficult for him to connect the act of looking at the thermometer with interacting with the wood burner. However, neither of these problems could be best solved by adding new constraints, and so were not part of this project.

To help him complete the tasks, the author gave him verbal feedback, not dissimilar to the type shown on screen. It had to be more frequent and more direct than would have appeared on screen, and without it, it is very unlikely that he would have completed the simulation.

4. Evaluation

4.1.Description and method

To determine whether indeed the hypothesis that '*it possible to model a user's progress through everyday tasks and evaluate the progress in real time*' holds, it was evaluated by a domain expert. The domain expert explored the virtual environment completing a number of tasks. Through the actions they made in pursuing that completion, various items of feedback was generated. The domain expert was able to compare this feedback with the feedback they expected from their knowledge of the constraints. After that the domain expert examined the generated user model to check that any breaches of constraints or lack there-of was recorded. The tasks that were selected were different from the tasks from the pilot study. They were selected to make every constraint relevant. For example, if there were no constraints involving the washing machine, then there is would be no need for a constraint regarding crouching. The six tasks were:

Tasks
When it rains bring in the washing.
When the washing machine finishes, hang out the white dress.
If the temperature drops below 18 Degrees, turn on the wood burner
At 5.57pm turn on the radio
After you listen to the racing results [on the radio] feed the fish
At 6.00pm turn on the Kettle

Table 6: Tasks presentment in the final evaluation

4.2.Results

All constraints were satisfied or violated as expected, and these results were recorded faithfully by the user model. All the feedback messages displayed their expected messages, and there were no bugs or crashes in the code. However there were some issues with the tasks that were demonstrated. Two of the tasks did not function as expected, and so the simulation could not be accurately completed. Several other recommendations were also made.

At some points the feedback actually led to the domain expert making more errors. In this situation, the user was alerted that they should be doing one of several tasks, and told all the tasks currently available. When the user completed the lowest priority of these tasks, they breached the constraint that they should be doing the most high priority tasks. This led to the recommendation that feedback messages should only suggest the single most important task right now.

Another area of confusion was when two constraints were violated separately, but in quick succession. Time-based constraints always waited 20 seconds between displaying feedback, but if any two other constraints were violated in quick succession the first feedback message would be replaced by the second message, and the first message would be lost to the user.

There were issues with inventory-based tasks, that only the name of one relevant item would be considered a 'goal'. For example in the task bring in clothes from the line, the item was clothesline, but the items that actually required interaction were dress (green), dress (white), dress (orange), and dress (blue).

Since the expert evaluation, all these criticisms have been addressed and implemented. It would seem that now the hypothesis holds that: *it possible to model a user's progress through everyday tasks and evaluate the progress in real time.*

4.3.Discussion

Ideally to test the hypothesis, we would have conducted a large scale experiment, with a statistically significant number of stroke patients, each of whom would have been recorded completing tasks in the virtual environment. After that, a domain expert would compare the recordings with the user models generated to asses if the user models were accurate. This however proved to be infeasible with the time and resources available. Let us discuss in this section the validity of the results given the assessment conducted.

While it would be infeasible for the domain expert to test every possible outcome, their test did involve every constraint in some way and they were in working order. They had also previously examined the constraints to assess any flaws in their coverage. As well as conducting a typical usage

scenario covering all of the constrains, they covered several emergent scenarios where they were concerned they could be issues.

The pilot study was similar to a typical use case scenario. During the Pilot study, the domain expert also examined the feedback along with the user model, and made several recommendations. As the system has now been used by a stroke survivor, more credibility has been added, as we can evaluate if it would be used paradigmatically differently by a stroke patient and test likely emergent scenarios from the outcome.

In a perfect evaluation, we would prefer to evaluate every single task possible to test if some combination of tasks creates some emergent outcome. This was not achievable within the COSC486 timeframe. However, the majority of objects that can be interacted with, were used in some way in the expert evaluation (7/11 interact-able objects, excluding inventory objects). Because all objects in the virtual environment inherit from two super classes, we have basis to believe that any new tasks in the future would produce an unexpected and negative interaction with the user model.

It is unfortunate that it could only be evaluated on a small scale. However, we must consider: the determinism of the system, that we have evaluated it in a real life scenario, that it was evaluated by an expert, the intractable nature of exhaustive testing, and the task and constraint coverage. When we take these matters into consideration, it would appear that the level of evaluation was sufficient. Therefore we can hold the hypothesis '*it possible to model a user's progress through everyday tasks and evaluate the progress in real time.'* to be true.

5. Conclusion

The virtual environment was created as part of a project to improve users' prospective memory. This necessitated a user model as well as constructive feedback for the users. To accomplish this, the author has used a constraint-based modelling approach to add user models. Constraint-based modelling is a popular technique in intelligent tutoring systems; it compares the user's current state with that of a set of constraints necessary for a correct solution. The user model is a collection of information the system has regarding the user's knowledge and learning strategies.

For the project he author has crafted 15 constraints. These are evaluated by a program also of his design which returns feedback to the virtual environment. There were a number of key design decisions outlined in section 3, including submissions, priority and developing constraints. In addition to this the author developed an editor for constraints to increase maintainability. Details of the constraints and their scripting language were also related in Section 3. A case study was conducted to assess the usability of the system for a stroke survivor.

To evaluate the project's hypothesis - *It possible to model a user's progress through everyday tasks and evaluate the progress in real time* – an expert evaluation was conducted. The expert arrived at the conclusion that the project does produce the expected feedback and record the user model accurately. The experts also made a number of suggestions for improving the clarity of feedback, and for improving the system overall. These have since been implemented. Section 4 also discussed the validity of the findings. An expert evaluation is sufficient for the project, particularly considering time and resource considerations.

The primary outcomes of the project are the feedback system and the constraints which will be used for the larger trial and also proving of the hypothesis. We can clearly say from the information in this report, that *It possible to model a user's progress through everyday tasks and evaluate the progress in real time.*

5.1. Future Direction

There are a number of areas that could be explored to extend this research. To increase the usefulness of the feedback/user model, the user model could help the virtual environment decide which tasks to present the user. Another opportunity to make the feedback more helpful for stroke patients would be to have a text to language system read out the feedback. If there was a larger evaluation of the effectiveness of the system in the future that would lend more validity to the system. Eventually it is likely that more constraints will have to be added to the model to capture additional edge cases.

Most other intelligent tutoring systems have in some form a mechanism for choosing the next task based on the user model. This is an extremely useful function as it ensures that users are always improving their weaknesses without wasting time using skills that they have already mastered. In this project, that was not as necessary, as tasks are related to another experiment currently being set up involving perspective memory. However for other future experiments, tasks selection would be important. Constraints already have ontology information, so if tasks were to also have their required skills labelled, it would be trivial to add this functionality in its most basic form.

Many stoke patient suffer from some form of communication disability, such as dyspraxia or aphasia, as a result of their injury. If the system was able to read out the feedback, as well as present it on screen, this would broaden the audience of stroke patients for whom it could be helpful. Even for users without communication disabilities, it would still be worthwhile, as it would be one less thing on display on screen requiring the user's attention and distracting them from the object they should be looking at.

Section 4 of this report described how a larger study could be conducted. If this were to happen, it would give much greater validity to the system. While this would be useful to ensure its ongoing use, it probably would not lead to an improved outcome for end users.

Additional constraints are part of the maintenance phase of this software project. As the project is dynamic to an extent, and impossible to exhaustively test, eventually it may be the case that other behaviours are employed by the users than what was anticipated. If these behaviours or patterns are detrimental to the user's performance, it may be necessary to add more constraints to the system to ensure that the user receives appropriate feedback informing them as to what behaviour they should be using instead.

There are four main areas to be approached for future work on this project. Two of these, task selection and synthesized speech, are improvements which would improve the functionality and usefulness of the program. Additional constraints are likely to be a necessary part of maintenance as usage of the program increases, while the last area is in regards to ensuring the validity of these techniques.

6. Bibliography

1. **Groot, Y., et al., et al.** Prospective memory functioning in people with and without brain injury. *Journal of the International Neuropsychological Society.* 2002, Vol. 8, pp. 645-654.

2. **Mathias, J.L. and Mansfield, K.M.** Prospective and declarative memory problems following moderate and severe traumatic brain injury. *Brain Injury.* 2005, Vol. 19, 4, pp. 271-282.

3. **Maujean, A, Shum, D and McQueen, R.** Effect of Cognitive Demand on Prospective Memory in Individuals with Traumatic Brain Injury. *Brain Impairment.* 2003, Vol. 4, 2, pp. 135-145.

4. **McDaniel, M, et al., et al.** Delaying execution of intentions: Overcoming the costs of interruptions. *Applied Cognitive Psychology.* 2004, Vol. 18, pp. 533-547.

5. *Virtual reality in neuroscience research and therapy.* **Bohil, CJ., Alicea, B. and Biocca, FA.** 2011, Nature Reviews Neuroscience, Vol. 12, pp. 752-762.

6. *A virtual reality apartment as a measure of medication skills in patients with schizophrenia: a pilot study.* **Kurtz, M., et al., et al.** 2007, Schizophrenia Bulletin, Vol. 33, pp. 1162-1170.

7. *Virtual Reality in Psychotherapy: Review. CyberPsychology & Behaviour.* **Riva, G.** 3, 2005, Vol. 8, pp. 220-230.

8. *Validity of a Virtual Environment for Stroke Rehabilitation.* **Edmans, J., et al., et al.** 11, 2006, Stroke, Vol. 37, pp. 2770-2775.

9. *A Strategy for Computer-Assisted Mental Practice in Stroke Rehabilitation.* **Gaggioli, A., Meneghini, A., Morganti, F., Alcaniz, M., and Riva, G.** 4, 2006, Neurorehabilitation and Neural Repair, Vol. 20.

10. **Kinsella, G, Ong, B and Tucker, J.** Traumatic Brain Injury and Prospective Memory in a Virtual Shopping Trip Task: Does It Matter Who Generates the Prospective Memory Target? *Brain Impairment.* 2009, Vol. 10, 1, pp. 45-51.

11. *Use of Virtual Reality Tasks to Assess Prospective Memory: Applicability and Evidence.* **Knight, R and Titov, N.** 1, 2009, Brain Impairment, Vol. 10, pp. 3-13.

12. **Ohlsson, S.** Learning from performance errors. *Psychological Review.* 1996, Vol. 103, 2, pp. 241-262.

13. *Fifteen years of Constraint Based Tutors: What we have achieved and where we are going.* **Mitrovic, A.** 1-2, 2012, User Modelling and User-Adapted Interaction, Vol. 22, pp. 39-72.

14. *The LISP tutor.* **Anderson, JR. and Reiser, BJ.** 4, 1985, Byte, Vol. 10.

15. **Ohlsson, S.** Constraint-Based Student Modelling. [book auth.] J Geer and G McCalla. *Student Modelling: The Key to Indicidualized Knowledge-Based Instruction.* s.l. : Springer-Verlag, 1994.

16. *Intelligent tutors for all: Constraint- Based Modelling methodology, systems and authoring.* **Mitrovic. A, Martin. B, Suraweera. P.** 2007.

17. **Sandberg, J.A.C.** 1987. The Third Internaiontal Conference on Artifical Intelligence and Education. pp. 51-53.

18. **Mitrovic, A, Koedinger, K and Martin, B.** A Comparative Analysis of Cognitive Tutoring and Constraint-Based Modeling. [book auth.] p Brusilovsky, A Corbett and Fiorella de Rosis. *Lecture Notes in Computer Science.* Berlin : Springer Berlin / Heidelberg, 2003, Vol. 2702, pp. 313-322.

19. *Evaluation of a Constraint-Based Tutor for a Database Language.* **Mitrovic, A. and Ohlsson, S.** 10, 1999, International Journal of Artificial Intelligence in Education, pp. 238-256.

20. *Tailoring feedback by Correcting Student Answers.* **Mitrovic, A. and Martin, B.** 2000. Intelligent Tutoring Systems. pp. 383-392.

21. *What Brings Intentions to Mind? An In Situ Study of Prospective Memory.* **Sellen, A.J., et al., et al.** 5, London : Erlbaum, 1997, Vol. 4, pp. 483-507.

22. **Shum, D, Valentine, M and Cutmore, T.** Performance of Individuals with Severe Long-Term Traumatic Brain Injury on Time-, Event-, and Activity-Based Prospective Memory Tasks. *Journal of clinical and experimental Neuropsychology.* 1999, Vol. 21, 1, pp. 49-58.

23. *Creating and Evaluating Problem Templates for Problem Generation Within the Context of Stroke Cognitive Rehabilitation.* **Ogden, S.** 2012, Unpublished report.

24. **Kodaganallur, V.** A Comparison of Model-Tracing and Constraint-Based Intelligent Tutoring Paradigms. *International Journal of Artificial Intelligence in Education.* 2005, Vol. 15, 2, pp. 117-144.

25. **Brewer, G., et al., et al.** A comparison of activity-based to event-based prospective memory. *Applied Cognitive Psychology.* 2011, Vol. 25, 4, pp. 632-640.

26. *Constraint-Based Tutors: a Success Story.* **Mitrovic, A., et al., et al.** 2001, Engineering of Intelligent Systems., pp. 931-940.

27. *A Critique of Kodaganallur, Weitz and Rosenthal, "A comparison of Model-Tracing and Constraint-Based Intelligent Tutoring Paradigms.* **Mitrovic, A. and Ohlsson, S.** 3, 2006, Artifical Intelligence in Education, Vol. 16, pp. 277-289.

7. Appendix

7.1.Setup instructions

7.1.1. Server

The server requires quite a number of files to operate. First there is feedback generator.jar. This is the main program that receives the actions and returns the feedback. It requires that in the same directory is house.txt, usermodel.dat, and constraints.dat. House.txt is a graphical representation of the house. Each line consists of a line number, room name, and the line numbers of all the rooms it connects to. Usermodel.dat is the serialized format of an object containing all the user models.

Constratints.dat is a serialised array containing all the constraints in the system. Neither user model nor constraints are human readable.

Also on the server side, it is required that there is a folder at c:\data\reports. This is the directory where the user models will be exported to in CSV format.

7.1.2. User

The user's computer requires the network helper if it is to be executed on a different computer to the server. If the network helper does not have a conf.ini file in the same directory, it will assume that it is operating over loopback. By creating a conf.ini file, you can specify the IP address of the server you would like to connect to.

The environment itself requires all its support files. It will also write a file with the name of the user in it. To connect the unity environment to the network helper and then server, push 'n' with the network helper and server already in operation. If this is successful, the feedback message 'connected' will be displayed.

7.2. Link to code listing

The code listing is too long to reproduce here. It can be found in a repository on the ICTG server, in the Brain-Injury-Tutor\SourceUnSafe\Scott\Feedback folder.